Hyperdrive:
A Continuous Delivery Report

LaunchDarkly → SLEUTH

Continuous delivery is the process of releasing new code to quality assurance for testing on a rapid, continuous basis.

Everyone wants to deliver better software without sacrificing speed or safety.

And yet, the velocity of releasing software remains a relatively subjective experience for each organization. For instance, it's very easy to say you're performing continuous delivery (CD), but what that actually looks like in practice differs depending on the company.

To get a better feel for the state of continuous delivery and how teams are delivering software, LaunchDarkly and Sleuth partnered with DZone to survey software developers, architects, site reliability engineers, platform engineers, and other IT professionals.

For the purposes of this report, we're defining continuous delivery as the process of releasing new code to quality assurance for testing on a rapid, continuous basis.

Our questions aimed to gain a stronger understanding of the motivations driving the adoption of continuous delivery, metrics being tracked, use of feature flags, and more.

KEY FINDING 1

Reasons for adopting Continuous Delivery

It's been said that the c-suite loves continuous delivery more than developers do.

We wanted to understand why continuous delivery is attractive to software professionals in general, and specific job roles in particular, so we asked and then segmented by job role.

The top two reasons for adopting continuous delivery (increased speed of feature delivery; shortened development cycles) are related to velocity from a developer's point of view. Other goals that represent the same improvements — but from project management/ops perspectives, e.g. increased release frequency and reduced deployment error rate — are ranked lower overall.*

Reason	Score
Increased speed of feature delivery	1732
Shortened development cycles	1504
Increased release frequency	1364
Improved developer/team flow/productivity	1195
Reduced complexity of development cycle	1127
Reduced deployment error rate	903
Reduced overhead costs	897
Reduced time to complete QA feedback loops	887
Reduced maintenance cost	847
Reduction in number of bugs post deployment	839
Reduced mean time to discovery (MTTD)	632
Reduced error budget	537

Scores are computed by weighted rank: if, for a given response, answer A is ranked highest out of N answer options, then A's score is incremented by N, while if answer B is ranked second highest out of N answer options then B's score is incremented by N-1.

> **Continuous Integration and Continuous Delivery needs to be balanced with safety mechanisms as moving too quickly can lead to unintended consequences.**

The main reasons companies implement continuous integration/continuous delivery is for speed. But speed needs to be balanced with safety mechanisms, as moving too quickly can lead to unintended consequences. The safety mechanisms rank higher when examined by role.

Developers rank “reduced deployment error rate” seventh (score: 747), SREs rank “reduced deployment error rate” sixth (score: 24) and platform engineers rank “reduced deployment error rate” 12th and last (score: 17). These varying rankings suggest different social stakes for CI/CD among roles.

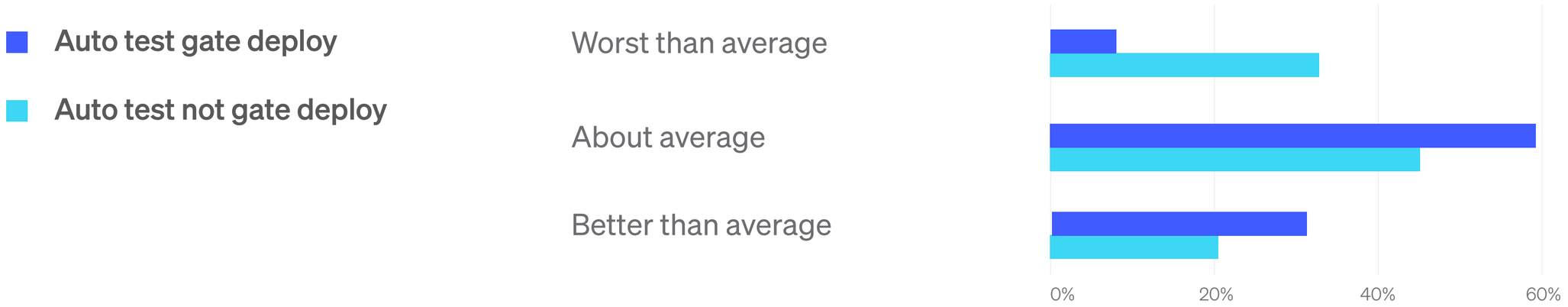
If you’re looking to implement CI/CD in your organization, the justification to get buy-in across the organization should vary by role.

Actual vs Desired Deployment Frequency

Although many factors influence optimal software release frequency—many of which could not be reasonably smoothed by a single objective distribution—we wanted to see if there might be an overall subjective answer to the question of which is the best frequency.

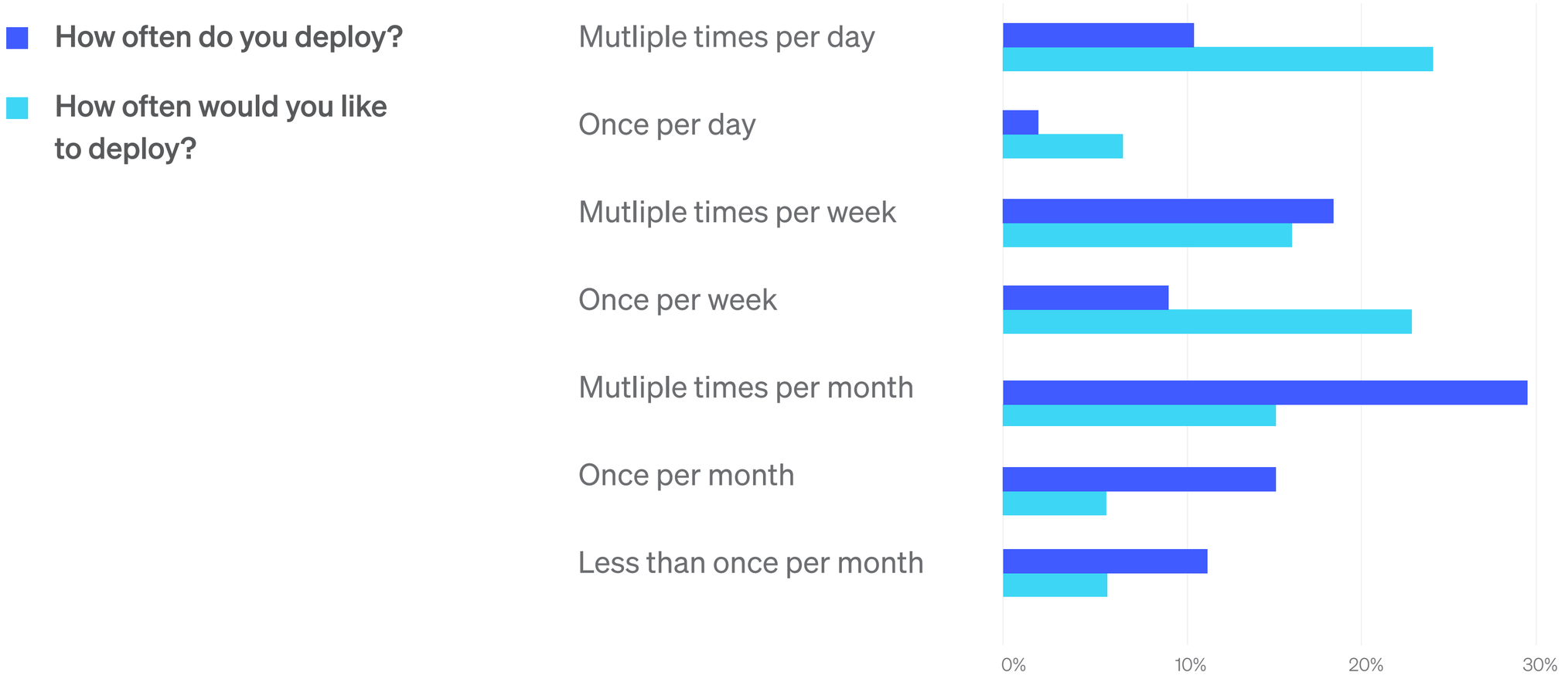
So we asked how often are folks deploying, if that's at a favorable rate, and how they would measure themselves against other teams.

Compared to other development teams, I believe our deployment process is:



There is a desire to shift deployment frequency "right" to more frequent deployments. Those who are deploying multiple times per month would like to shift towards deploying multiple times per week. And the respondents deploying once per day want to shift to deploying multiple times per day.

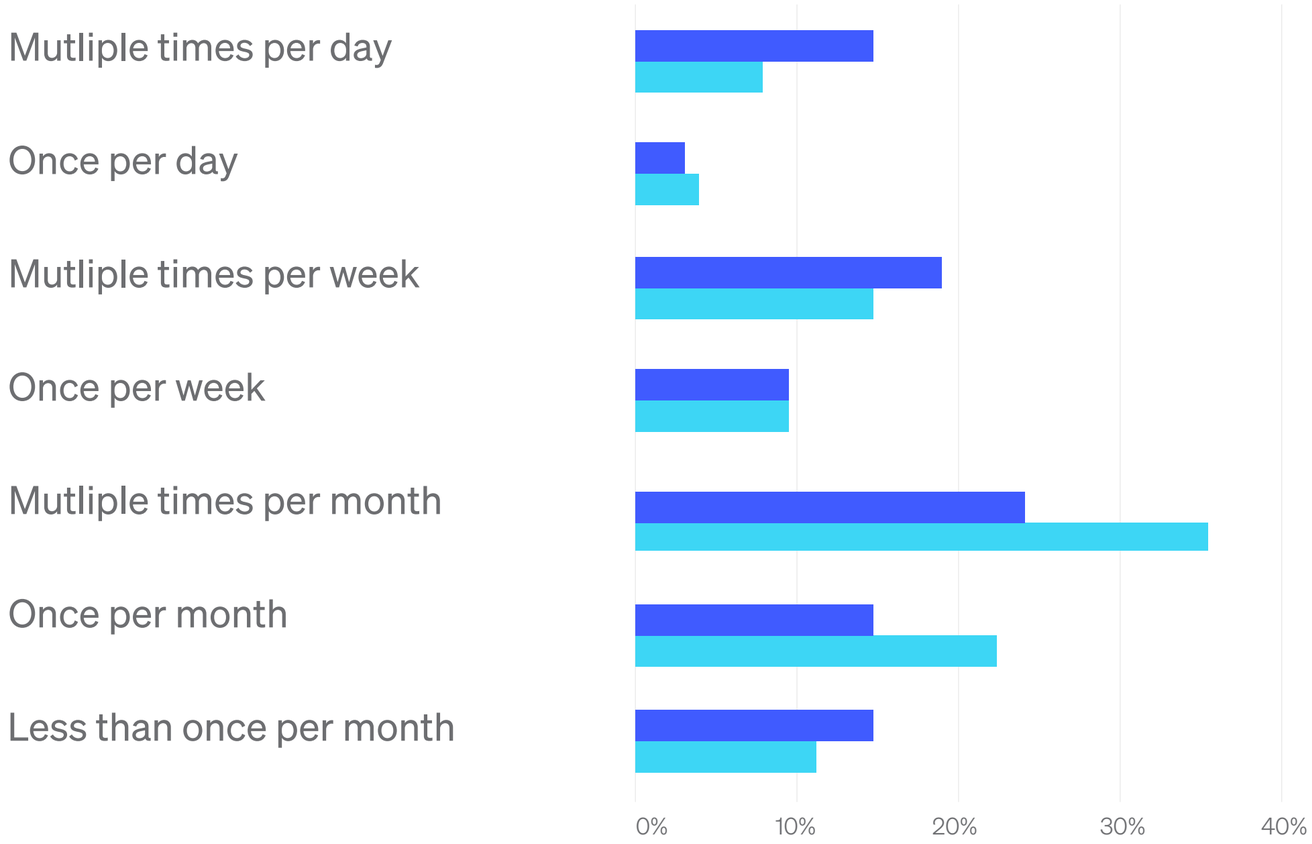
Over 25% of respondents want to be deploying multiple times per day. However, this may be a case where in theory this sounds good, but in practice may be problematic, depending on whether or not the right processes to support this pace of deployment are in place. For instance, 13% of those who release multiple times per day believe their current rate of deployment is too fast.



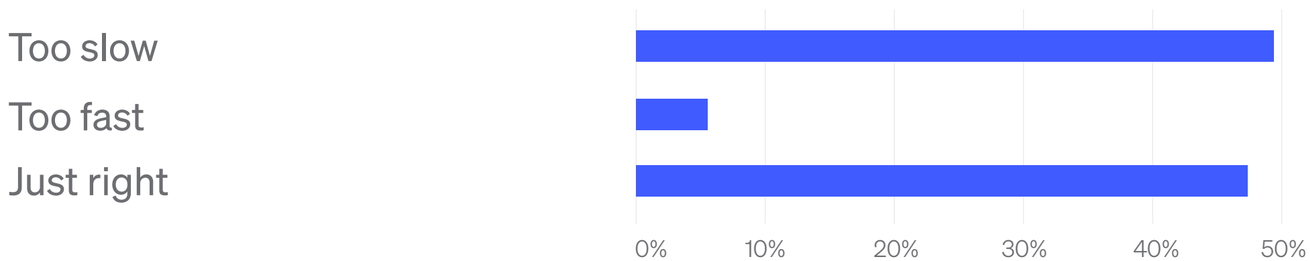
Feature flag users

Feature flag non-users

How often do you deploy?

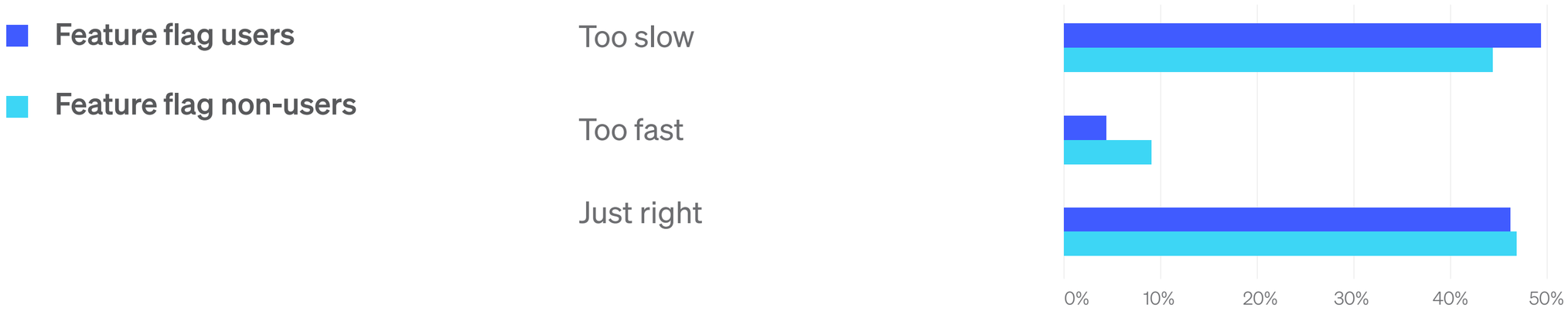


Few software professionals think they are deploying too fast. Satisfaction with the rate of deployment appears almost evenly split between too slow (48.3%) and just right (46.3%).



This suggests that, in the grand scheme of things, and in proportion to software professionals' good judgment, software should be delivered more rapidly—i.e. delivery ought to be more continuous.

How often do you deploy?



Among the few respondents who think that their deployments are currently too fast, significantly more not-primarily-developers like CI/CD engineers, platform engineers, SREs, and others think they are currently deploying too fast, while devs think they are deploying too slow.

10 out of 11 respondents who said that they are moving too fast have a higher rate of incidents associated with deployments.

One reason for not releasing faster is often the fear of broken code or incidents. In fact, 10 out of the 11 respondents who said that they are moving too fast have a higher rate of incidents associated with deployments. Contrast this with the respondents who said their deployment rates were just right, 47 out of 94 said they rarely have an incident related to a deployment.

While actual causes of rollbacks and incidents are far too many and complex to be inferred from this correlation, the data suggests that software professionals significantly associate deployment failure with prematurity in general. This result confirms the intuition that getting deployment rate right may be a good way to reduce rollbacks and incidents.

Teams use Metrics to Measure Continuous Delivery

Among the top five metrics teams used to measure their continuous delivery practice, four of them are the “Big Four” metrics popularized in the book “Accelerate”:

1. Deployment frequency
2. Change failure rate
3. Change lead time
4. MTTR

Deployment frequency, in particular, is the No. 1 metric for obvious reasons.

This isn't a surprise, as these metrics have been scientifically proven to affect software delivery performance through research conducted by DORA over the past six years.

Deployment frequency, in particular, is the No. 1 metric for obvious reasons.

The remaining top metric, and No. 2 overall, is production downtime during deployment. This speaks volumes to the need to increase speed without sacrificing safety and reliability.

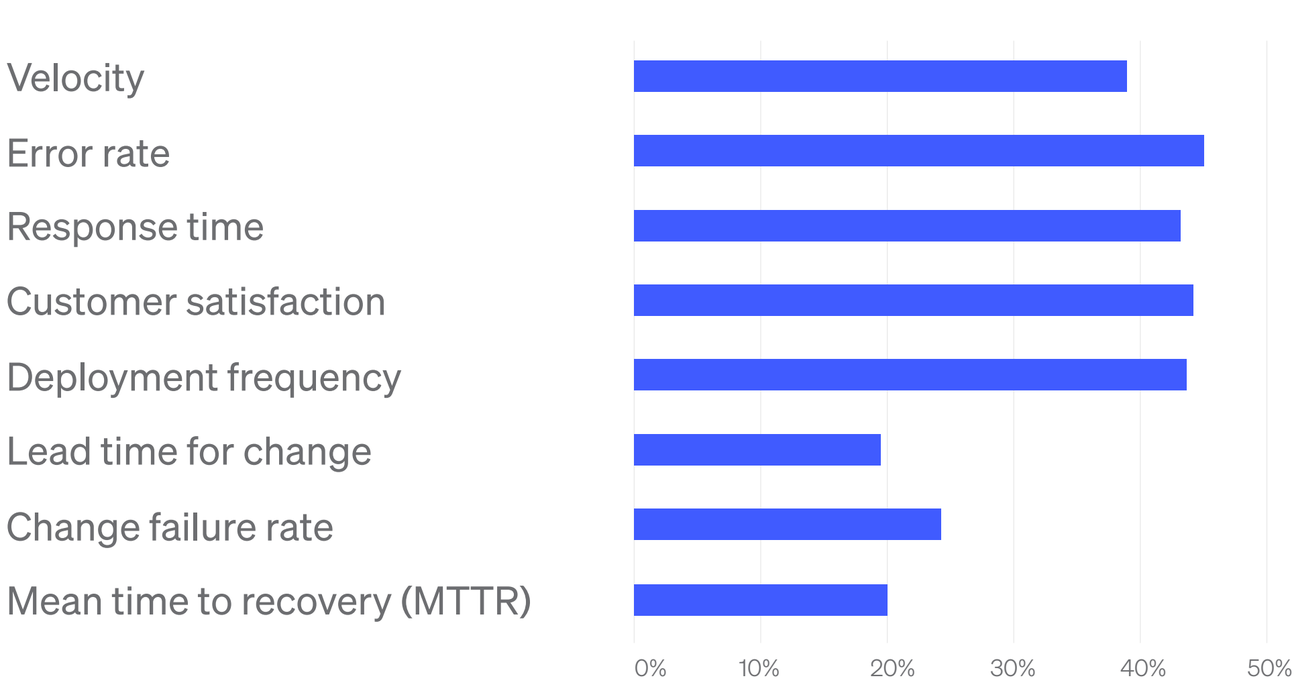
Item	Overall Rank	Rank Distribution
Deployment frequency	1	
Production downtime during deployment	2	
Lead time	3	
Change failure rate	4	
MTTR(mean time recovery)	5	
Regression test duration	6	
Error rate	7	
MTTD (mean time to discovery)	8	
Absolute number of bugs	9	
Error budget	10	

Tactically, when it comes to measuring success of deployments, four metrics are top of mind among teams surveyed at almost equal measure:

1. Error rate
2. Customer satisfaction
3. Deployment frequency
4. Response time

One thing these metrics have in common is they should be relatively easy to access. Customer satisfaction (CSAT) and response time metrics are likely existing measurements teams can use right away, even if they're only lagging indicators of deployment success.

What metrics do you track related to your deploy?



KEY FINDING 2

More room for automation

As complex business logic evolves, manual testing can clog release pipelines and is prone to errors.

And while realistic automated unit testing of well-designed software is, in theory, reasonably straightforward, it can also be labor-intensive. Certain kinds of automated testing—like UI, load testing, and security testing—may require compute resources in addition to the application's own runtime host.

> **Deploying software is a technical decision. Releasing software is a business decison.**

Further, when Agile/JIT-style methodologies are used, unit-level tests may evolve over the course of development, increasing the pre-release risk of unexpectedly failing tests, missing test scenarios, and a late-discovered need for downstream refactoring.

We know it is essential to automate as much testing, provisioning, and mitigation facilitation as possible in order to undiscretify delivery. But we wanted to know how software professionals actually do gate production deployments.

Gating of Deployment by Automated Tests

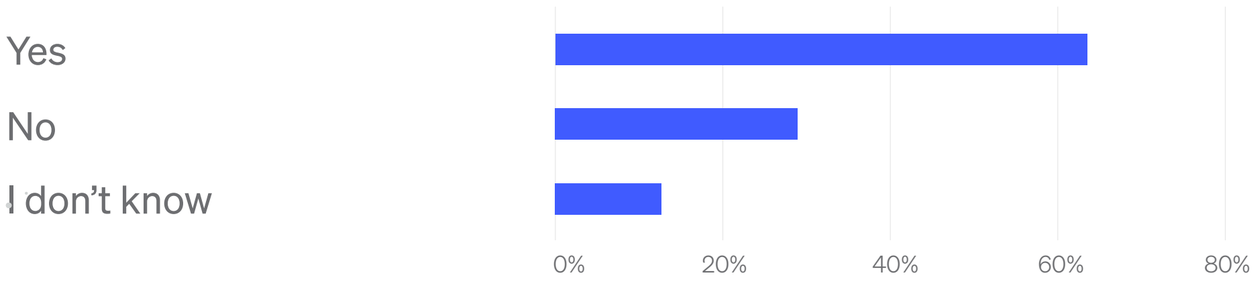
There is a distinction between deploying and releasing code. Deploying software is a technical decision, involving testing and verifying things are working correctly. You can test in local environments, staging environments, and in production.

Releasing software, on the other hand, is a business decision. And software should not be released without testing. Release means customers are interacting with the software, and if something goes wrong there are business implications.

While it's not possible to test every scenario, some degree of unit or integration testing is typically needed before deploying and releasing software. Given that testing is an important discipline, it was surprising that only 63% of respondents indicated that automated tests gate deploys.

It's possible that the 24% who deploy code without automated tests are testing that code in production behind feature flags or in some other capacity.

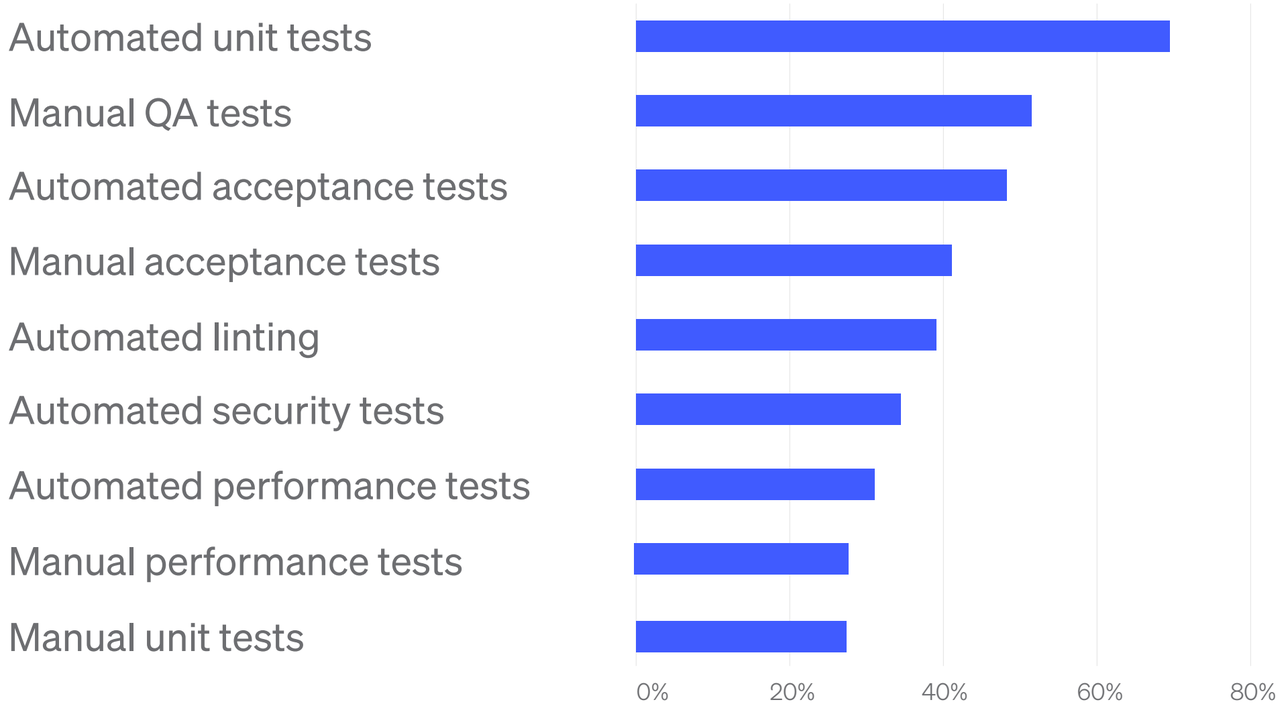
Do automated tests gate the deploys?



Tests Run Before Deploy

Aside from automated unit tests–which are the most fundamental–the most-commonly run tests before deployment are manual QA. Manual testing is far from the ideal of continuous delivery. The fact that a (small) majority runs manual QA tests before deployment suggests that, in general, the industry is far from achieving the testing infrastructure required for CD.

What tests do you currently run before deploy?



Blockers to Automating Production Deployments

The top three blockers to automating production deployments are ensuring:

1. There is a robust code review process
2. Unit test coverage is > 75%
3. Pre-production and production environments are in sync

Check all things that MUST be true (i.e. join by Boolean AND) in order to automate production deployments.

Production & pre-production evironments are provisioned by the same code.

Code review process is robust.

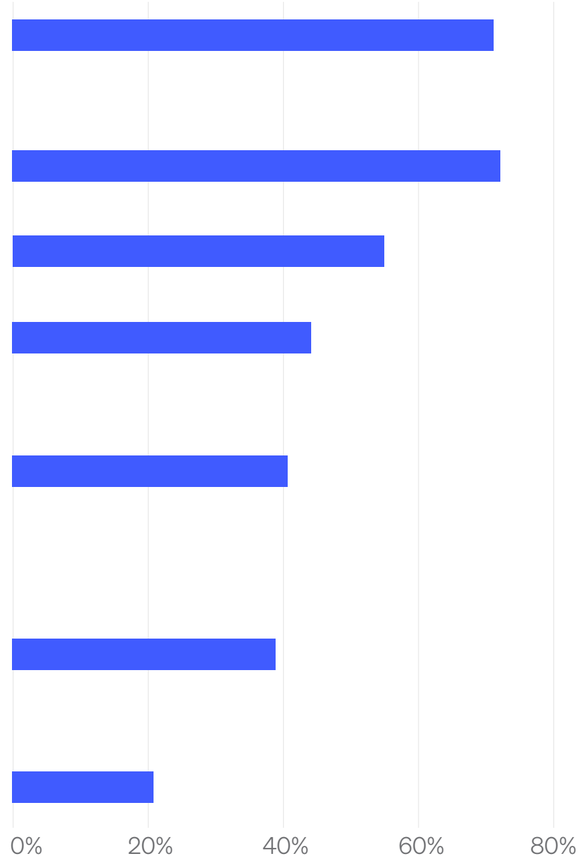
Unit test coverage is >75%

All whitelisted user paths are covered by automated UI tests.

There is an auditable approval process to promote changes from pre-production to production.

Crucial features are behind feature flags and can be turned off without a new deployment.

Unit test coverage is >75%



* Source

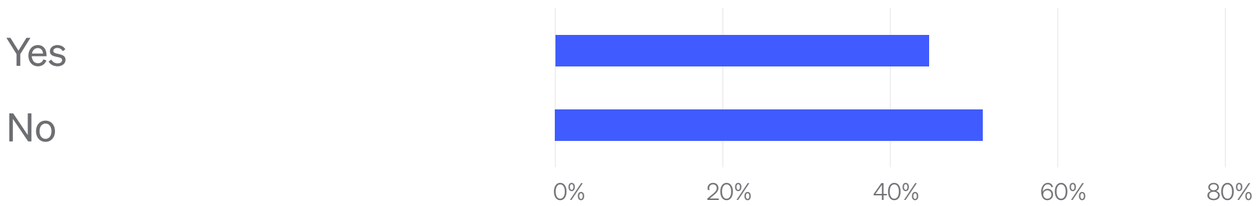
KEY FINDING 3

Feature flags are in early days

One process companies implement to deploy more frequently is feature flagging.

Feature flags are a way to control who can see a feature. With feature flags, code can be deployed and not released to all users.

Are you currently using flags to enable selective feature toggling without redeployment (i.e. feature flags)?



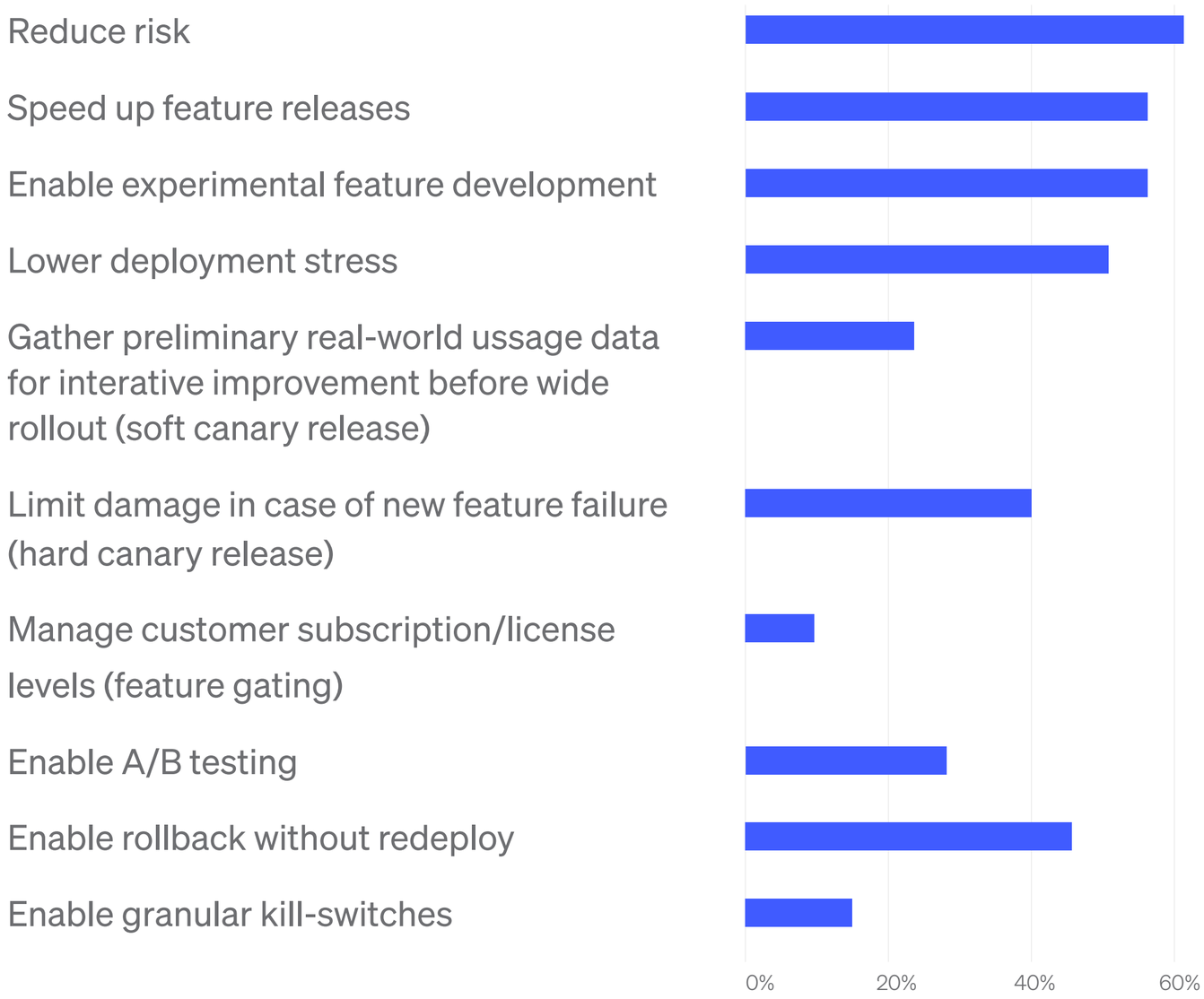
While some teams are using feature flags to speed up deploys and reduce risks, the majority (55%) of teams are not using feature flags. And for those teams that have incorporated feature flags as part of their development processes, it is a new addition within the last year.

Feature flags are used in many scenarios—including release management, experimentation, and for operational purposes. The primary reasons software professionals are using feature flags align with the two primary reasons for implementing CD: reducing risk and increasing the rate of deployments.

The top five responses were reducing risk, speeding up feature releases, enabling experimental feature development, lowering deployment stress (this is a risk mitigation issue and we'll have more on this later), and enabling rollbacks without deploys.

Flags make it possible for companies to move safely at speed. When something goes wrong after a deployment, a feature can be toggled off in a short period of time with no need to redeploy code.

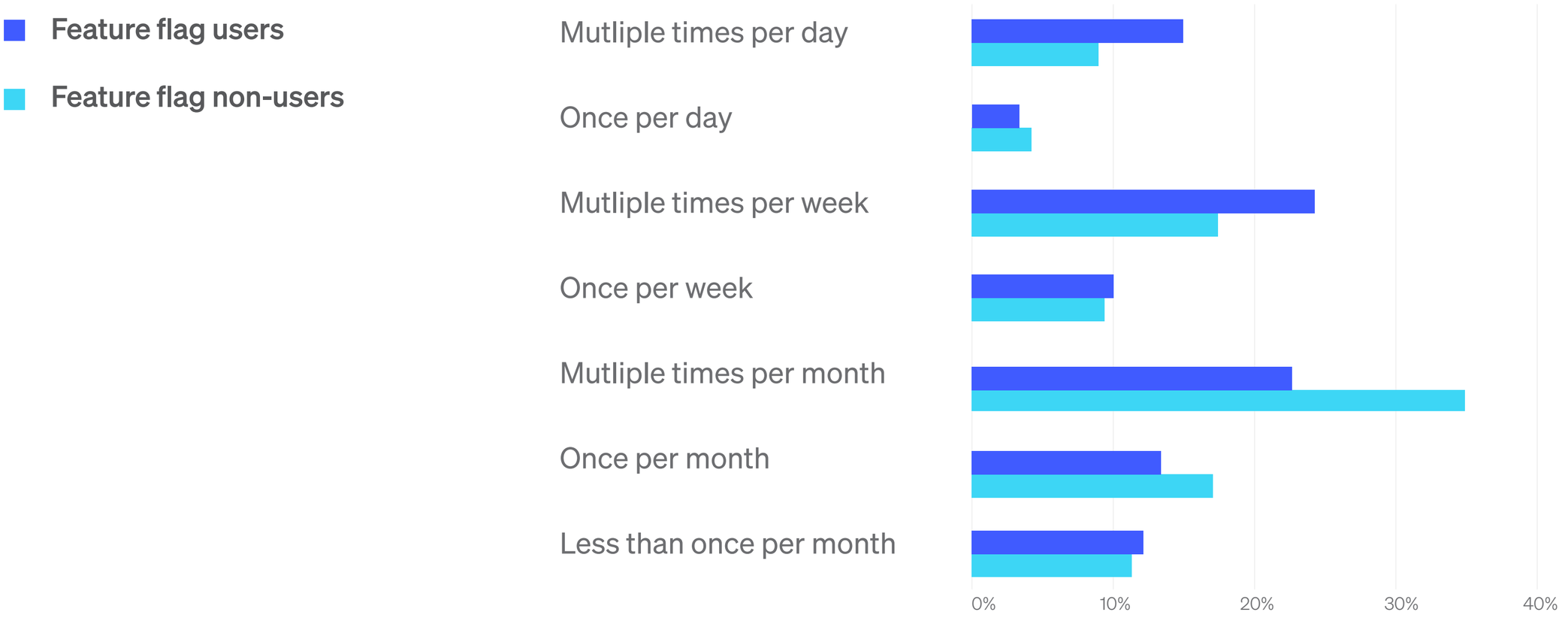
What are your top reasons for using feature flags?



Impact of Feature Flags

Since feature flags facilitate a more aggressive release pipeline via both feature-addition and risk-reduction, we supposed that the use of feature flags would correlate positively with other CD maturity indicators. This turned out to strongly be the case across a wide range of objective and subjective metrics.

How often do you deploy?

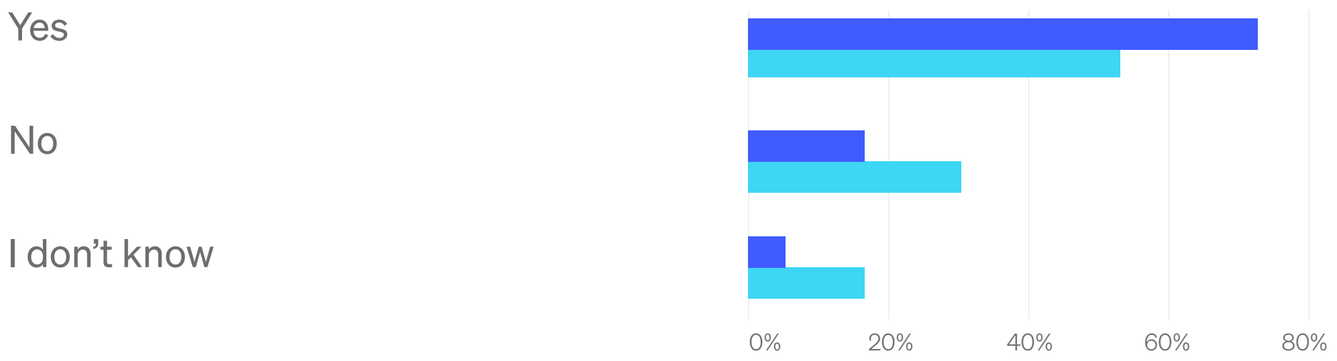


> **Respondents who use feature flags are significantly more likely to release multiple times per week or multiple times per day.**

Feature flag users

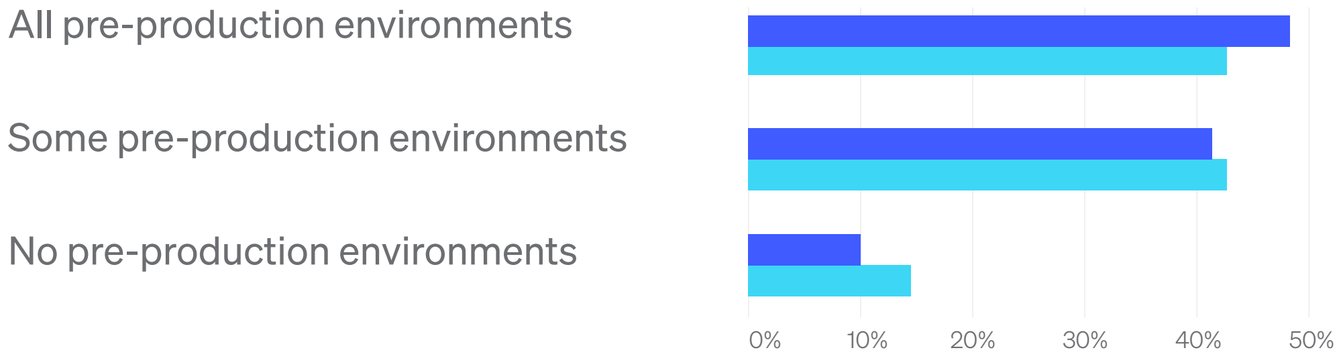
Feature flag non-users

Do automated tests gate the deploys?



> **Respondents who use feature flags are more likely to have automated tests gate deploys.**

We automated provisioning and deployment for

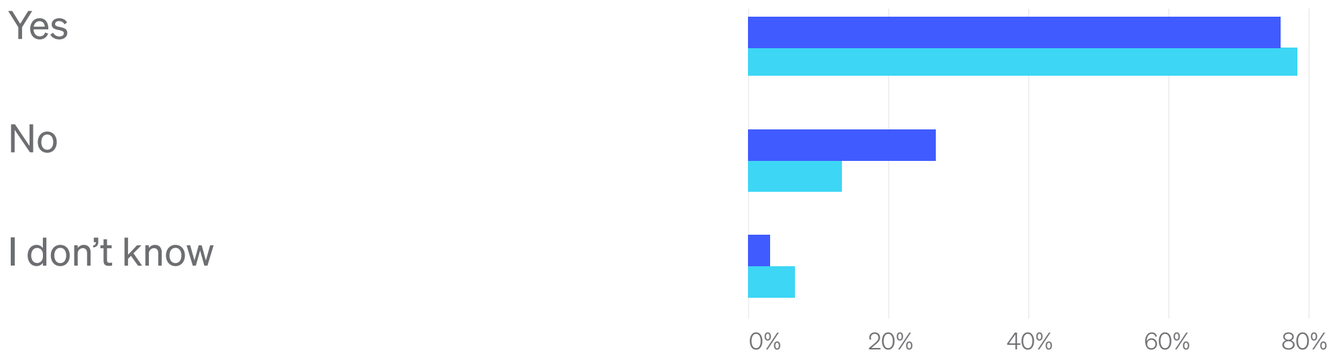


> **Respondents who use feature flags are more likely to automate provisioning and deployment for all pre-production environments.**

Feature flag users

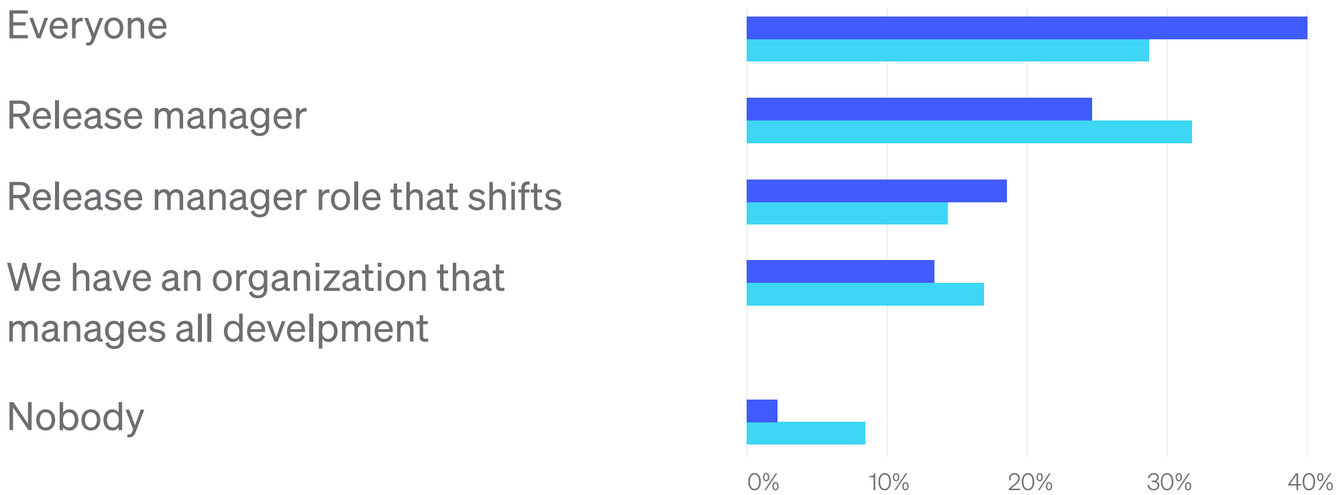
Feature flag non-users

Do your deployments to production requre any manual steps?



> **Respondents who use feature flags are less likely to require manual steps for production deployment.**

Who helps coordinate and monitor deployments for the team?

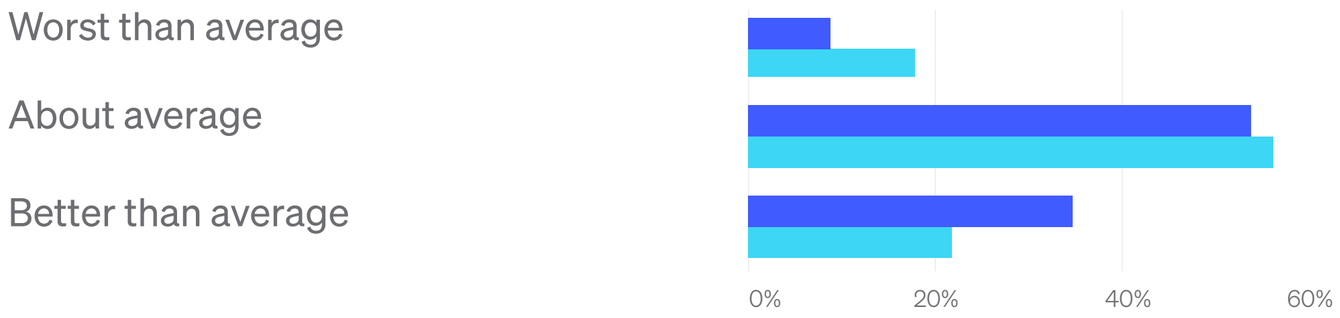


> **Respondents who use feature flags are significantly more likely to involve all team members in deployments.**

Feature flag users

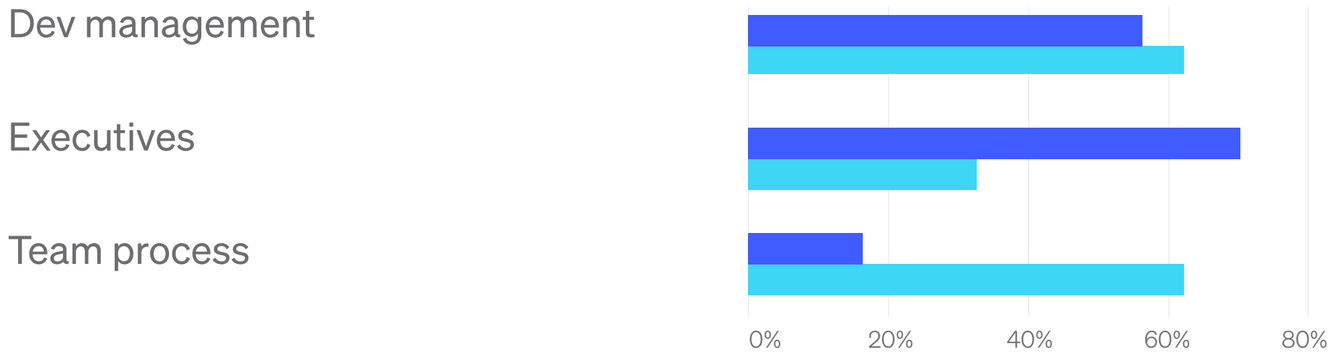
Feature flag non-users

Compared to other development teams, I believe our deployment process is:



> **Respondents who use feature flags are more likely to consider their deployment process better than average.**

What do you think is causing your organization to fall behind?



> **Respondents who use feature flags are far less likely to blame team process for organization-level deployment issues.**

I would describe my team's culture as...

- Feature flag users
- Feature flag non-users

Respondents who use feature flags are more likely to describe their team's culture as "excellence-oriented," "craftsmanship pride," or "dominated by engineers," and less likely to describe their team's culture as "chaotic" or "aimed at avoiding criticism."

KEY FINDING 4

The Human Impact

Anybody that has been involved in software releases knows that deployments cause stress and anxiety.

The worry over whether things will go smoothly or if people will be pulled away from dinner, sleep, or vacations is real.

All deployments are potentially stressful, but not all deployments cause anxiety.

When something goes wrong, people are impacted, whether it is the ops engineer who received the alert, the developer that needs to write code, the customer support team that fields emails or calls from upset customers, or the social media team that has to provide messaging. Software deployments impact humans, and we wanted to look at the extent of this impact.

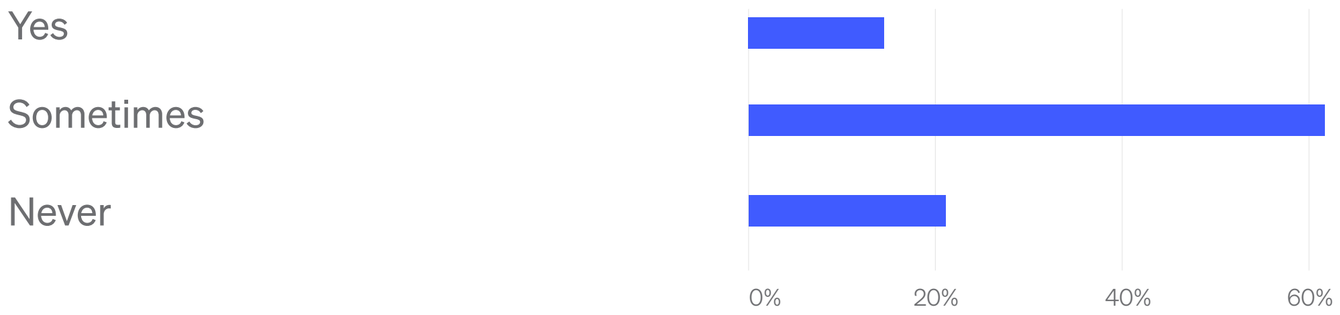
For example, does CI/CD improve not only the software delivery process but also the lives of the people delivering and supporting the software?

Prevalence of Deployment Anxiety

All deployments are potentially stressful, but not all deployments cause anxiety.

Stress is a change that causes physical, emotional, or psychological strain. Anxiety is a feeling of worry or unease about an imminent event, or something with an uncertain outcome. You can experience stress without experiencing anxiety. Experiencing too much stress is not good and can eventually lead to anxiety or other physical or mental health issues that will seriously impact performance at work and beyond.

Do you experience deployment anxiety?



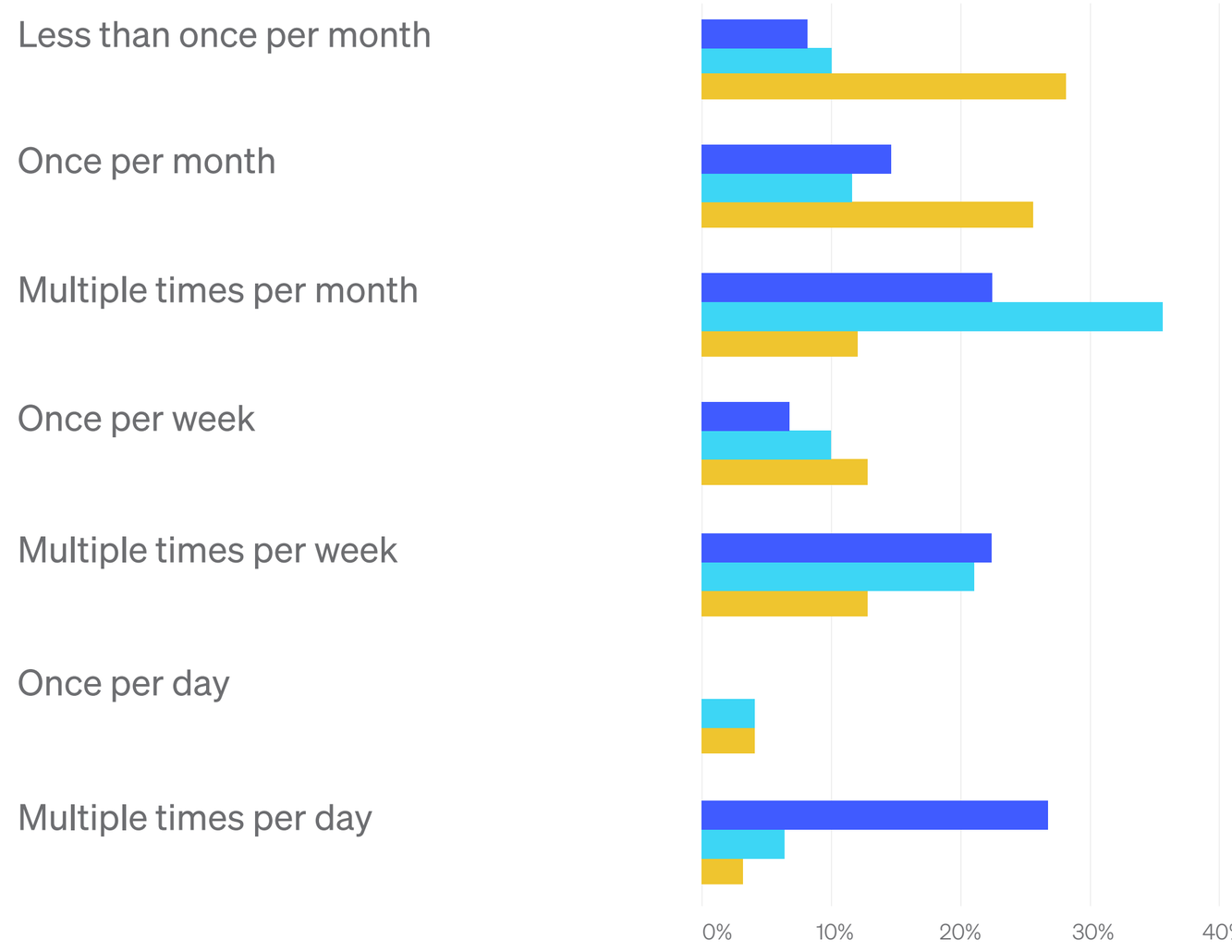
When we are talking about anxiety here, we do not mean in the clinical sense defined by the "*Diagnostic and Statistical Manual of Mental Disorders*" published by the American Psychiatric Association (e.g. Generalized Anxiety Disorder). Anxiety disorders are a serious subject and should be treated by a mental health professional. In this case, we are referring to anxiety as an emotional response that is not necessarily related to an anxiety disorder.

Correlates of Deployment Anxiety

Since many of our findings on correlates of deployment anxiety are clear and actionable. Results can be listed especially concisely, with limited speculation on causal mechanisms. Results were striking enough that we plan additional analysis in future publications.

How often do you deploy?

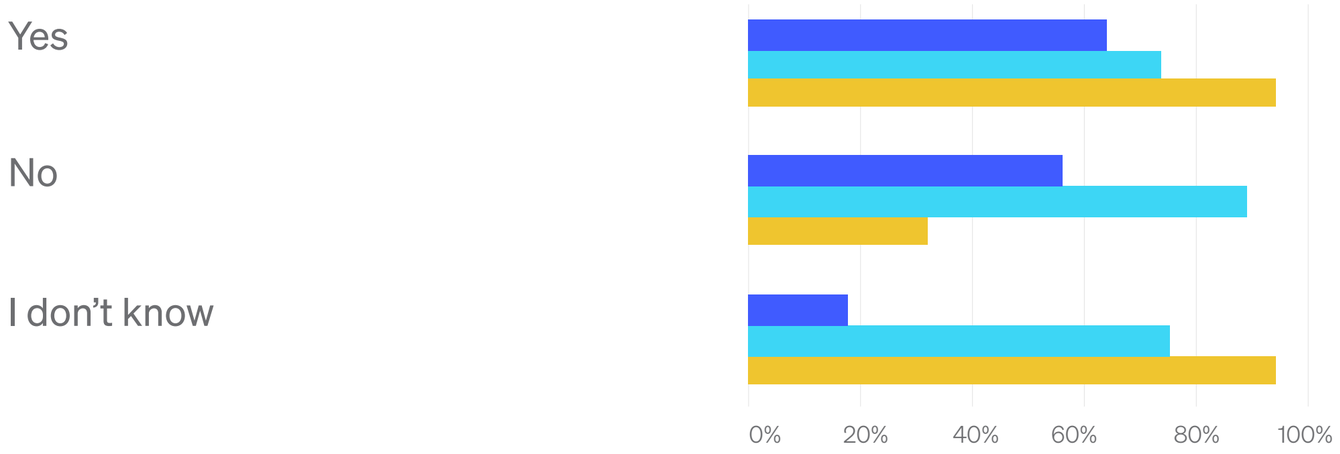
- Never experience deployment anxiety
- Sometimes experience deployment anxiety
- Always experience deployment anxiety



> **Deployment frequency strongly inversely correlates with deployment anxiety.**

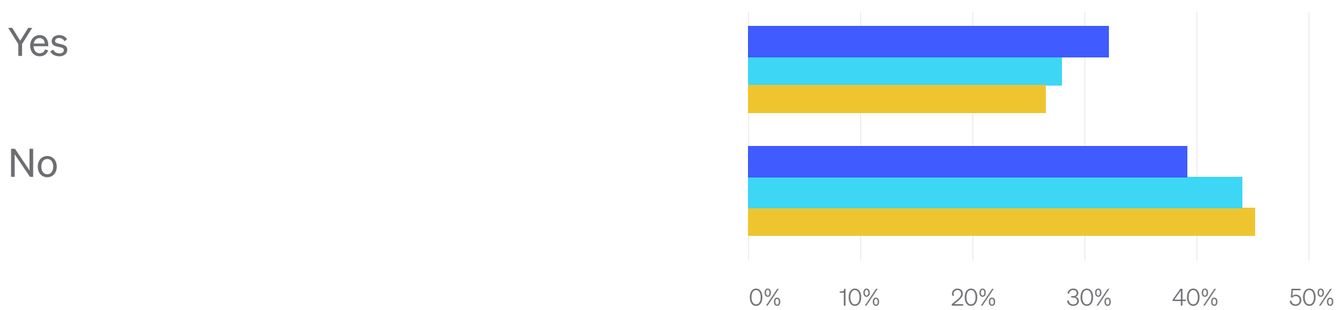
- Never experience deployment anxiety
- Sometimes experience deployment anxiety
- Always experience deployment anxiety

Do your deployments production require any manual steps?



> **Manual deployment steps required for production deploy correlate positively with deployment anxiety.**

Are you currently using flags to enable selective feature toggling without redeployment (i.e. feature flags)?



> **Use of feature flags mildly correlates inversely with deployment anxiety.**

Check ALL things that must be true (i.e. join by Boolean AND) in order to automate production deployments.

- Never experience deployment anxiety
- Sometimes experience deployment anxiety
- Always experience deployment anxiety

Unit test coverage is 100%

Unity test coverage is >75%

Unity test coverage is >50%

Code review process is robust

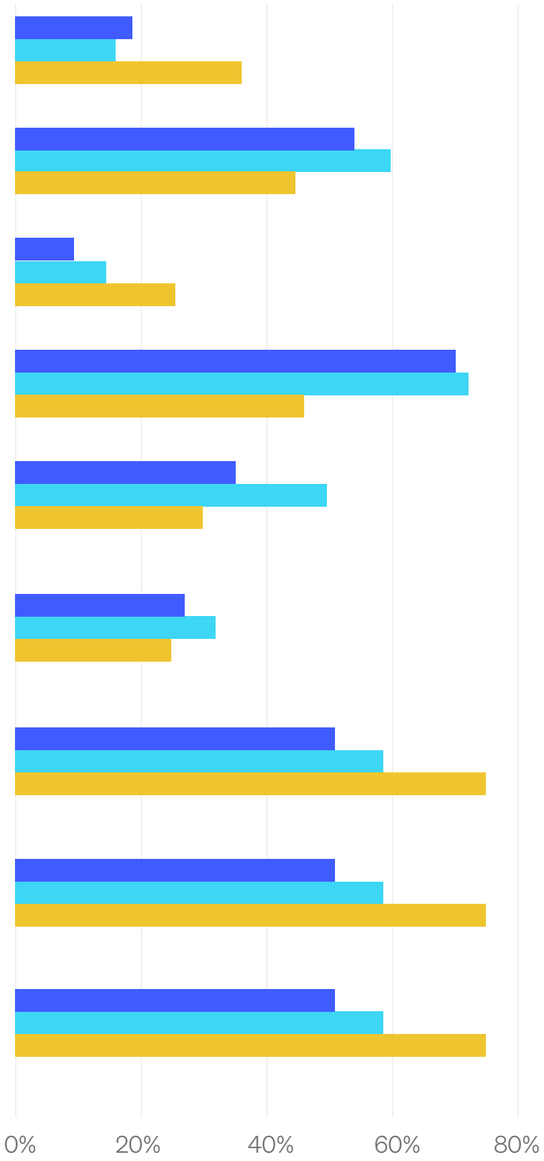
All whitelisted user paths are covered by automated UI tests

“Most” (a fuzzy definition) user paths are covered by automated UI tests

Realistic load tests are performed on every build

Production & pre-production environments are provisioned by the same code

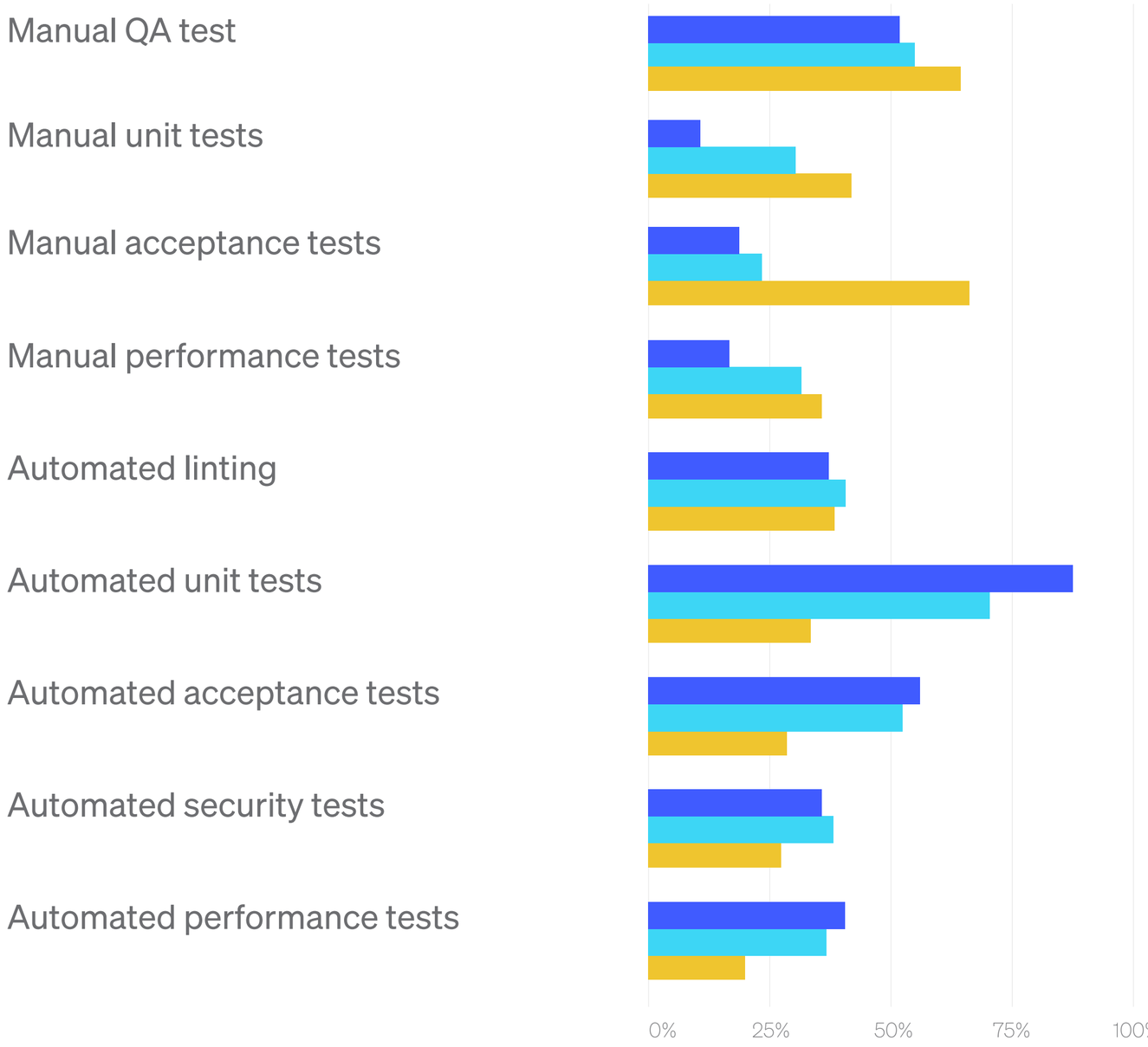
Crucial features are called behind feature and can be turned off without a new deployment



> **100% unit test coverage correlates with higher deployment anxiety.**

- Never experience deployment anxiety
- Sometimes experience deployment anxiety
- Always experience deployment anxiety

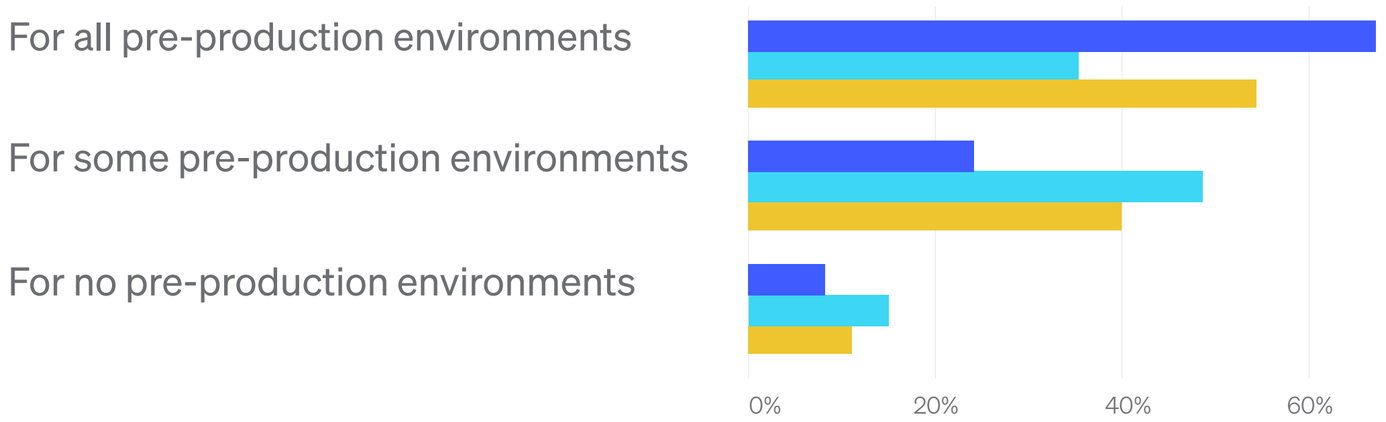
What tests do you currently run before deploy?



Manual QA and acceptance tests positively correlate, and automated unit and acceptance tests inversely correlate, with deployment anxiety.

- Never experience deployment anxiety
- Sometimes experience deployment anxiety
- Always experience deployment anxiety

We've automated provisioning and deployment



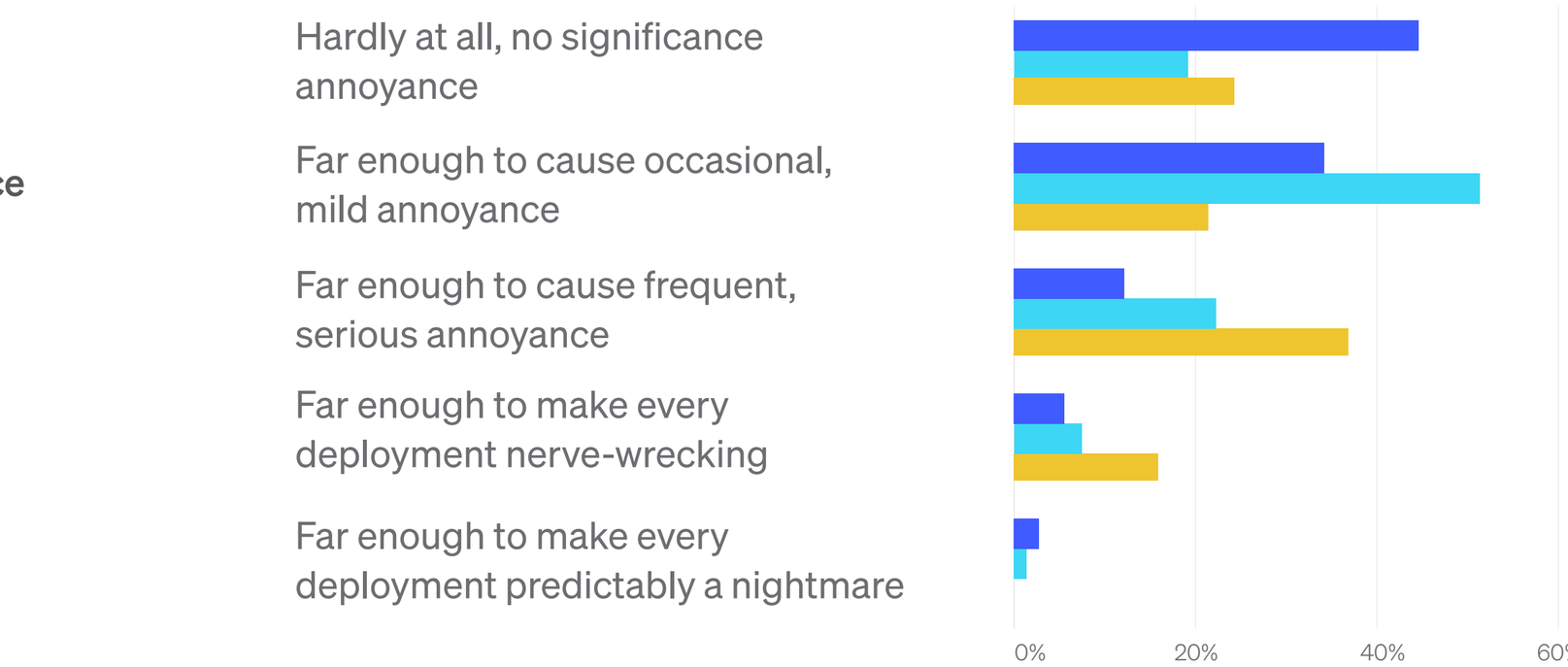
Automated provisioning and deployment prevalence across environments inversely correlates with deployment anxiety.

Observation

This finding is consistent with our earlier finding on prerequisites for production deploy, but the present finding is stronger insofar as this question quantifies over environments along the release pipeline.

- Never experience deployment anxiety
- Sometimes experience deployment anxiety
- Always experience deployment anxiety

How far apart (how much drift) do your different environments get on a regular basis?



> **Environment drift strongly correlates with deployment anxiety.**

I would describe my team's culture as (check all that apply)

Many social dysfunctions correlate positively with deployment anxiety.

Respondent Demographics

Geographic breakdown

United States	95
Canada	17
Western Europe	51
Eastern Europe	32
Australia	5
New Zealand	3

Role breakdown

Lead/Directory/VP of Engineering or Software Development	70
Software developer/engineer	88
CI/CD engineer	15
Platform engineer	10
Site reliability engineer	6
Other	14

Methods

We created a survey and distributed it to an audience of software professionals in the United States, Canada, Western Europe, Eastern Europe, Australia, and New Zealand. Question formats included multiple choice, free response, and ranking. Survey links were distributed via email to an opt-in subscriber list, popups on DZone.com, and short articles soliciting survey responses posted in a web portal focusing on devops-related topics. The survey was opened and closed in March 2021. The survey recorded 203 complete responses.

Conclusion

Hopefully the results of this report can shed light on the greater landscape of continuous delivery, as well as how things are working at your organization. For more updates on these findings in the future, visit 100deploysaday.com.